Persistent, Reliable, Decentralised File System - DFS

Derrick Robertson, Paul McKee, Cefn Hoile

BTExact, {derrick.robertson@bt.com, paul.mckee@bt.com, cefn.hoile@bt.com}

Abstract: This paper describes the architecture of a global, distributed, reliable data store. The system is based on a decentralised peer to peer network and uses cryptographic checksums to ensure validity and security and erasure codes to distribute fragments across the network to a set of peers whose attributes match the parameters specified by the user's preferences.

1. Introduction.

Moving into the 21st century has seen the increase in both shared resources and digital data. Resources such as bandwidth, hard-disk space and computer cycles can be shared through common distributed techniques such as GRID, NAS (Network attached storage) and SAN (Storage area networking) technologies but the amount of digital information required to be stored and accessed by individuals and large companies is doubling per month. This combined knowledge pushes us away from traditional data storage methods towards more dynamic, ubiquitous archival techniques evolved from the starting point presented by Rabin's algorithm [1]. The IDA (Information dispersal algorithm) is an early demonstration of file fragmentation. This means only storing a fragment of the file on a number of machines, moving away from the mirror site ideology where the complete image of the file is available on all machines.

The functions that are sought in a data archival structure are:

1. Persistence: The data is always available until the author decides to remove the archive from the network.

2. Availability: Data is accessible 5 9's of the time.

3. Performance: System able to offer a suitable quality of service.

4. Security: Data should not be able to be modified or accessed by other users of the network unless authorised. This provides an element of privacy and integrity.

5. Resilience: The system is able to recover after peer and network failures and can suitably adjust to new peers joining network.

Several systems have already tried to encapsulate these ideas into peer to peer storage infrastructures. Such systems allow a peer to represent both server and client roles. These systems include the likes of Silverback [2], SFS [3], JungleMonkey [4], DistribNet [5] and OceanStore [6], but in some aspects each has failed to address all the requirements listed above. For example, Jungle Monkey only makes use of fragmentation to transfer the host file to the client so this can be seen as reducing availability. Distribnet fails to maintain user security as peers can read cached information therefore not providing substantial privacy. The initial scope for the creation of DFS was to construct a system, which would comply with all aforementioned requirements. With these properties incorporated its uses can be extended to disaster recovery, large file hosting and local file sharing.

2. Decentralised File System (DFS)

The system allows users to delegate the storage of files to participating machines in a fully decentralised peer network. The benefit of this system is it provides greater availability and resilience at equal redundancy levels of storing mirror images. This feature of the system allows the end user to reduce the large costs associated with generating and storing large amounts of data such as purchasing SAN/NAS devices or larger hard-drives. As the system is fully decentralised, the user can host files without depending on a specific machine in the system as information is fragmented and stored on peer's computers. This allows the user to be able to recover the files even if the users own computer is not functioning. The DFS has been designed with pluggable modules so that each user can be using different modules but still retain the same basic functions mentioned below. These different features can include the identifier for the file and file fragmentation parameters. The system has been designed, initially, on the basis of trust. The system can therefore be used in existing secure environments such as sub-nets where it is known that users will not intentionally abuse privileges granted. As such no mechanisms for monitoring and restricting of publishing have been included although to exist outside of a trusted relationship, this issue would have to be addressed. One solution would be the use of a marketplace scenario. As one peer hosts a fragment of a particular size, he is allowed to publish a file of determined size.

The methods available to the DFS are:

1. Publishing – publishes a file and stores it under a unique file identifier such as CRC.

2. Unpublishing – unpublishes a file from a given file identity.

3. Retrieval – retrieves a file from a given identity.

Through the use of only these three options, only the latest publication exists on the network structure. Since the main aim of the design of DFS was to minimise the amount of digital information whilst also providing reliability, the agents carrying the fragments of the any document published previously will terminate. It is possible to destroy indexing and leave fragments in the system. Possible solutions to this include a time to live mechanism being associated with the fragment or an agent (automatic process) to travel the peer network searching for obsolete fragments. The latter solution would be optimum to reach fragments that might not have received the termination request as out with the peer network at the time of the deletion command.

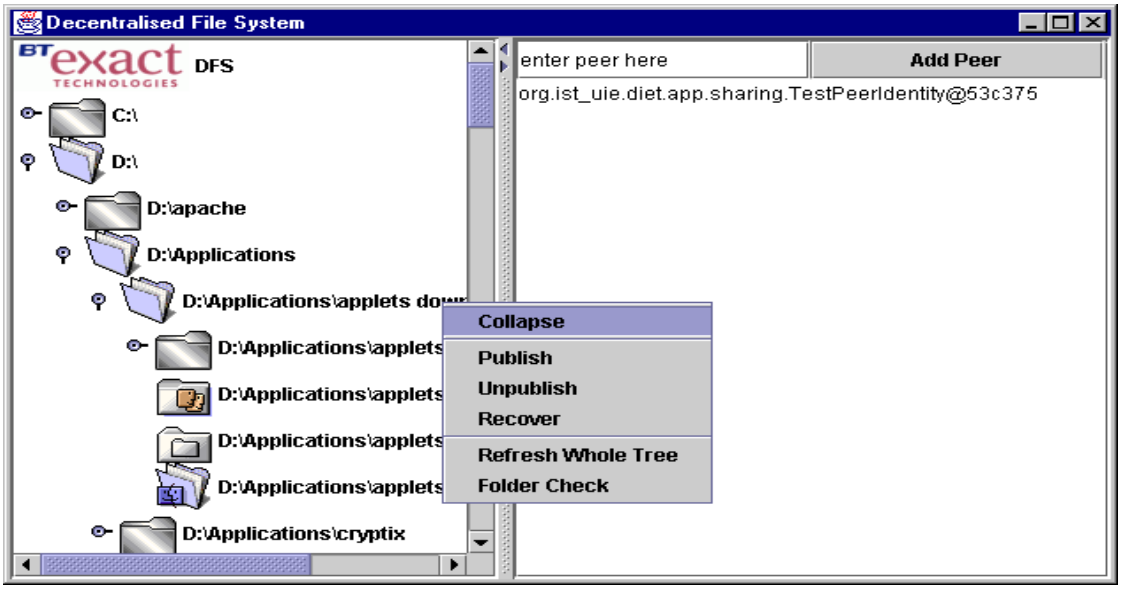

Figure 1 – DFS graphical user interface

The graphical user interface integrates the local file tree structure and the network file structure into one view so in an instance a user can determine the status of a file as shown in figure 1. There have been four circumstances identified for the existence of a file:

1. Local - exists solely on the local hard disk.

2. Network - exists solely on the peer network.

3. Identical - the copy found on the local file system is an exact copy of the image fragmented on the network

4. Updated - the file on the local file system has been updated since the file was published to the network.

With the system being able to relate between the local file system and the network file system this means that the user doesn't have the problem of remembering specific file identities. For the purpose of this implementation, the networked data is stored in a single flat file, which itself can be distributed within the peer network. This means that the local file system can be recovered in its entirety using only a single file identifier. This feature addresses the very important issue of usability and reduces the amount of interaction required from the user to maintain functionality.

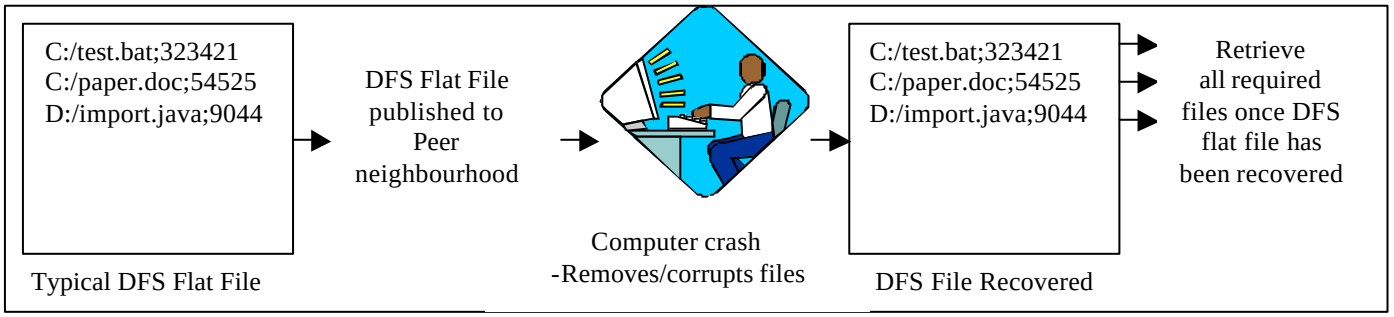

Figure 2: Local File System Back-up – recovered through knowledge of one file identifier

3. Security - File Identification

The implemented scheme for file identification is a 32-bit cryptographic checksum. This 4-byte number is generated on the contents of the file and is impossible to replicate the contents of a file to produce the same two checksums. In this way, the filename becomes obscured which means peers hosting the fragment have no idea what is contained within the fragment. Secondly, it enhances the security aspects of the system as if a fragment is tampered with and managed to recompile then the two checksums would not be identical. Another example is using a 160-bit SHA-1 hash where the probability of two objects having the same value would be 1 in 10^{20}[6].

4. Persistence - Erasure codes

To achieve this fragmentation, the system has moved away from complete image replication, towards the use of erasure codes. Through the use of these codes, high availability and persistence can be achieved without adding excessive amounts of redundant information to the system [7].

These erasure codes allow the file to be broken up into n blocks and encoded into kn fragments where k>1. The file can then be reassembled from any k fragments. This offers a significant advantage in a network of transient peers, since only k of the selected peers need to be available to allow file retrieval and no specific sub-groups need to be intact. Through the user's preference, the parameters of n and k are modified to achieve the appropriate degree of redundancy and reliability. The type of erasure code used in the system to transform the file is the Vandermonde FEC algorithm [8].

5. Performance

Although using erasure codes increases the availability and time a document can exist in the network, it creates further problems: The location of fragments and the retrieval of the fragment data. This is accomplished with the use of two further technologies. DIET – a lightweight, decentralised and scalable toolkit for mobile programs in peer to peer networks. This provides us with the capabilities of using agent based systems for constructing peer to peer neighbour in the small world sense and distributing fragments to selected peers environments [9]. SWAN – a system of self-organising indexes supporting pervasive access to dynamic resources in decentralised networks. This allows us to assign global identifiers to fragment agents so they can be easily located in an Euclidean based model [10].

Using these two systems, a further benefit is complete anonymity. This means that not only is the published file contents and name obscured from other peers but also the identity of the publishing and retrieving peer.

6. Conclusions and Further work

This paper gives an overview of the functions and architecture of the decentralised file system. It has shown how the authors have provided solutions to the problems related to distributed data storage especially in the areas of redundancy, persistence, conservation, availability and recovery and how these differs from the previously constructed systems. DFS is especially focused towards usability and this can be shown through the minimum amount of knowledge required by the user to publish/retrieve files located in the peer network and the interactions required to operate the graphical user interface.

References.

[1] Rabin, M., "Efficient Dispersal of information for security, load balancing and fault tolerance," Journal of the association for computing machinery, Vol. 36, No. 2, April 1989, pp 335-348

[2] Weatherspoon, H., Wells, C., Eaton, P., Kubiatowicz, D., Zhao, B., "Silverback: A global-scale archival system", U. C. Berkeley Technical Report: UCB//CSD-01-1139, March 2000 http://oceanstore.cs.berkeley.edu/publications

[3] Fu, K., Kaashoek, F., Mazieres, D., "Fast and secure distributed read-only file system", OSDI 2000, http://www.fs.net/

[4] Carra, A., Helder, D., Mukesh, A., "Jungle Monkey: Bulk File Transfer", Distributed Systems, EECS No. 589, 1998. http://www.junglemonkey.net/papers.html

[5] Atkinson, K., DistribNet, http://distribnet.sourceforge.net/design.txt

[6] Weatherspoon, H., Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., "Maintenance-free global data storage", IEEE Internet Computing, Vol. 5, No. 5 Sept/Oct 2001 http://oceanstore.cs.berkeley.edu/publications

[7] Weatherspoon, H., Kubiatowicz, J., "Erasure coding vs. replication: a quantitative comparison", IPTPS 2002, March 2002 http://oceanstore.cs.berkeley.edu/publications

[8] Luby, M., Mitzenmachery, M., Shkrollahiz, A., Spielmanx, D., "Practical loss-resilient codes", ITW, February 8-11, 1998. http://www.icsi.berkeley.edu/~luby/PAPERS/itw98.ps

[9] Hoile, C., Wang, F., Bonsma, E., Marrow, P., "Core specification and experiments in DIET: A decentralised ecosystem-inspired mobile agent system." http://lia.deis.unibo.it/confs/aamas2002/ix.jsp?key=accepted

[10] Bonsma, E., "Fully decentralised, scalable look-up in a network of peers using small world networks." Proc. 6th Multi Conf. on Systemics, Cybernetics and Informatics (SCI2002), Orlando, July 2002